AMENDMENTS TO THE SPECIFICATION

Please delete paragraph [0027] beginning at page 7, which starts with "Figure 4 is a".

Please replace paragraph [0028] with the following amended paragraph:

[0028] Figure [[6]] 5 is a code listing illustrating a portion of a serialized application in accordance with the present invention.

Please replace paragraph [0029] with the following amended paragraph:

[0029] Figure [[7A]] 6A is a source code listing of an exemplary portion of an application.

Please replace paragraph [0030] with the following amended paragraph:

[0030] Figure [[7B]] 6B is a source code listing defining a class included in the application of Figure 7A.

Please replace paragraph [0031] with the following amended paragraph:

[0031] Figure [[8]] 7 is a source code listing of objects serialized in accordance with the method of the present invention.

Please replace paragraph [0032] with the following amended paragraph:

[0032] Figure [[9]] 8 is a code listing illustrating a portion of classes functions serialized in accordance with the method of the present invention.

Please replace paragraph [0048] with the following amended paragraph:

[0048] The serialized representation may be stored as a text file, an XML file, or in another format, for later use by the rebuilder engine. Alternatively, the representation may be provided directly to a rebuilder engine. The dashed line between steps 310 and 312 in Figure [[4]] 3 indicates that the step of rebuilding is both optional, and separate in time from the optimizing process. The serialized representation created at 310 represents the network of objects in the memory of the application once the application has finished initialization.

Please replace paragraph [0057] with the following amended paragraph:

[0057] As noted above, the serialization step 310 may be called as a function of the runtime (built into the application), or a separate process to create the serialized representation in step 310. As a separate process, it inspects the memory state of the application. In a further embodiment, different runtime execution engines can perform steps 304 – 310, and to run the application in step 320. An example of this would be to simulate the behavior of the deployment platform in order to compute the initial application state. For example, the optimization process of Figure [[4]] 3 may be performed on any processing device shown in Figure 2 (including, for example, the application server 52). In one embodiment, it is performed on a separate processing device before the application is created by the rebuilder engine.

Please replace paragraph [0058] with the following amended paragraph:

[0058] Figure [[5]] 4 shows a method for performing the serialization process (step 310 of Figure [[4]] 3) in the present invention. Initially, at step 402, all object identifiers in the global scope are enumerated. At step 404, the first (or next) enumeration in the global scope is identified. At step 406, a determination is made as to whether the enumeration identified in step 404 identifies a media asset. If so, a pointer to the media asset is recorded in the serialized representation at step 408 and the method seeks the next enumeration at step 436. If the enumeration does not identify media at

step 406, then a determination of whether enumeration references a function is made at step 410. If a function is referenced, a determination of whether a closure is referenced is made at step 412. A closure is a pair of a function and a variable binding environment within which the function is executed. The function code is present in the object file. As explained in additional detail below, closures are transformed into objects which can be handled by the rebuilder. If a closure is enumerated, then the inner functions of the closure are transformed at step 414, and recorded in the serialization, and the next enumeration analyzed at step 436.

Please replace paragraph [0061] with the following amended paragraph:

[0061] A portion of one embodiment of a serialized representation created by the process described with respect to Figure [[5]] 4 is shown in Figure [[6]] 5. The example shown in Figure [[6]] 5 is generated from a Macromedia Flash runtime executing the output of a presentation server compiled a mark-up language description of a content application. In other embodiments of the invention, other types of source languages may be used. In this mark up language example, an application source file may contain at most one root element. In source application files, an element such as a canvas element defined in co-pending application serial no 10/092,010 defines properties of the global canvas, which is the root of the runtime view hierarchy; all other views are directly or indirectly contained within the canvas element.

Please replace paragraph [0062] with the following amended paragraph:

[0062] In the example of Figure [[6]] 5, the application is a calendar application. A root serialization identifier 600 is the first line of the description. A number of media object pointers 602, 604, 606 are shown. Each media object includes an ID (for example "M0") and attributes associated therewith between the "<...>". After all media assets are described, a root object 608 begins a listing of application objects and relations. The root object has a number of slots identified between brackets following the root object. Each slot may include a Boolean value "b", numerical value "n", serial

value "s", or object "o". A next object is identified at line 610. The object has a serialization ID of '1' and a name of "frameupdate." Two numerical slots are recorded, each with a name "n=" and numerical value "v=". A next object 612 is identified as "lzpreloader" having serialization ID of '2'. This object has three string slots, two numerical slots and a Boolean slot. The next object 614 is nested in the "lzpreloader" object 612. Like an object 612, it would have its own set of slots and nested objects. The method continues on until every object enumerated in the global scope is identified and serialized in this manner.

Please replace paragraph [0063] with the following amended paragraph:

[0063] An example of how classes are recorded in the serialized representation is shown in Figures ~~7A, 7B and 8~~ 6A, 6B and 7. Figure [[7A]] 6A is a source code portion of an application provided in a mark-up language such as that described in copending application serial number 10/092,010. This portion includes a header portion of the source code description and defines a class included at line 702 of "cal-button.lzx." The definition of the cal-button.lzx class is shown in Figure [[7B]] 6B. As shown in Figure [[7B]], the class "cal-button" has a number of attributes, including label, icon, divider, left cap, right cap, left insert, right insert, Y offset, and others.

Please replace paragraph [0064] with the following amended paragraph:

[0064] Figure [[8]] 7 shows a portion of a serialized application including objects created from the cal-button.lzx class. A first object 802 has a serialization ID 906 and has one string value slot of the name of the "cal-button" class as identified at line 804. The attributes (n="attrs") object 806 has series of attributes identified by SID 907 which are listed by lines identified at reference number 810. The values of each slot are identified along with the component of the object in accordance with the identifiers described above. Another instance of the cal-button class is identified at line 820 along with its attributes object 822 and events object 824. It should be understood that the two objects identified in Figure [[8]] 7 are but a portion of the serialized code beginning at the root level (shown

in Figure [[6]] 5) which continues on until each and every object in the global scope is identified and written in a serialized fashion.

Please replace paragraph [0065] with the following amended paragraph:

[0065] Figure [[9]] 8 shows an example of functions are identified using FIDs. Figure [[9]] 8 shows a number of functions for the class "animator" identified at line 902. For example, the "animator" class has an initialized function 904, a constructer function 906, and an init function 908, and a set start function 910. Each of the functions is associated with an FID as shown in each of the respective lines 904, 906, 908 and 910.

Please replace paragraph [0066] with the following amended paragraph:

[0066] In a further unique aspect of the present invention, the serialized representation is created in an output file, which as shown in Figures ~~6 through 9~~ 5 through 8, may be created in an extensible markup language (XML) format. It should be recognized that other formats may be utilized to store the serialized output of the present invention. The serialization process as described herein does not require that the interpreter know any property of any class, only what is in the class.

Please replace paragraph [0067] with the following amended paragraph:

[0067] The serialization process does not prioritize objects during the serialization process. In order to handle all types of runtimes, including a presentation renderer such as Flash wherein the content of the variable binding application cannot be introspected nor serialized, the serialization process uses single pass through the frozen application, and detaches each and every object as shown in Figure [[5]] 4.

AMENDMENTS TO THE DRAWINGS

The attached sheets of drawings include changes to Figures 5, 6, 7A, 7B, 8 and 9.

Figure 5 has been changed to Figure 4.

Figure 6 has been changed to Figure 5.

Figures 7A and 7B have been changed to Figure 6A and Figure 6B.

Figure 8 has been changed to Figure 7.

Figure 9 has been changed to Figure 8.